Методичка 4.1 - Сравнения и условные переходы

Компиляций программы на языке Си

Для демонстрации того, как работают условные выражения понадобится программа на языке Си, в которой будут использоваться различные виды условных выражений. Специально для этой цели была подготовлена программа prog-3.1.

Исходный код программы prog-3.1

```
#include <windows.h>

#pragma comment(lib,"user32.lib")

int main(int argc, char* argv[]) {

int tmp = 2;
char title[] = "Prog-3.1";

goto metka;
MessageBox(NULL, "Always Skipped", title, 0);
metka:

if (TRUE) { MessageBox(NULL, "Always TRUE", title, 0); }

if (FALSE) { MessageBox(NULL, "Always FALSE", title, 0); }

if (tmp < 5) { MessageBox(NULL, "tmp < 5", title, 0); }

if (tmp > 5) {
MessageBox(NULL, "tmp > 5", title, 0);
} else {
MessageBox(NULL, "tmp <= 5", title, 0);
}

if (tmp == 2) { MessageBox(NULL, "tmp == 2", title, 0); }

switch (tmp)
{
        case 1:
                MessageBox(NULL, "Case: 1", title, 0);
                break;
        case 2:
                MessageBox(NULL, "Case: 2", title, 0);
```

```
                break;
default:
MessageBox(NULL, "Case: default", title, 0);
break;
}
return 0;
}
```

Компиляция:

> cl prog-3.1.c -Od /Gs- /GS- /link /DYNAMICBASE:NO /NXCOMPAT:NO

Запуск:

> prog-3.1.exe

Мы пропустили сообщение “Always Skipped”. Далее мы видим сообщение “Always TRUE”. Затем мы пропускаем сообщение “Always FALSE”. Происходит сравнение переменной tmp со значением 5 и мы видим сообщение “tmp < 5”. Далее происходит снова сравнение с переменной tmp и мы получаем сообщение “tmp <= 5”. Затем происходит проверка на равенство - “tmp == 2”. И в конце мы видим сообщение, что был выбрал - “Case: 2”. На этом сообщении программа завершает свою работу.

Давайте откроем программу под отладчиком и узнаем, как реализованы сравнения и условные переходы на уровне инструкций процессора.

Анализ безусловного перехода

Переходим в начало секции кода (Ctrl + G, 0x00401000), где расположена единственная в программе функция main.

Ставим точку останова в начале функции (F2), чтобы при перезапуске программы сразу перейти в начало функции main.

Давайте немного изменим отображение переходов в окне дизассемблера.

Включим подсветку операторов перехода:

Кликаем правой кнопкой в области CPU - Appearance - Highlighting - Jumps and Calls

Затем попросим отладчик рисовать направление переходов в дизассемблированном листинге:

Debugging Options
CPU

Show direction of jumps
Show jump path
Show grayed
Show jumps

Теперь код читать стало легче.

В самом начале мы видим пролог функции, а также копирование локальных переменных из секции данных в стек через регистры EAX, ECX и DL.

Далее мы видим безусловный переход JMP. Это и есть реализация оператора GOTO. Мы перепрыгиваем через код с вызовом функции MessageBoxA.

Анализ условного перехода

Далее мы переходим к условному переходу.

В регистр ECX загружается значение 1. Далее мы видим инструкцию TEST ECX, ECX и условный переход JE.

Давайте разберемся, как это работает.

Оператор TEST работает также, как и оператор AND, т.е. выполняет логическую операцию &, но не сохраняет результат операции.

Как вы видите, инструкция TEST выполняется операцию & между одинаковыми значениями.

Вспомним таблицу истинности.

Нас интересует когда оба значения одинаковые, т.е. 0 & 0 = 0 и 1 & 1 = 1.

Если в регистре ECX будет значение 0, то результат будет 0, а если в регистре будет любое значение кроме нуля, то результат будет не 0.

Таким образом, можно сказать, что это проверка на нулевое значение в регистре.

Как вы помните, оператор JE ориентируется на значение флага ZF. Если значение в регистре ECX будет равно нулю, то переход будет осуществлен.

Если вы помните, то Zero Flag выставляется в 1, если последняя операция была равно нулю.

В нашем случае флаг ZF равен нулю, значит переход осуществлен не будет и мы попадаем на вызов функции MessageBox с текстом “Always TRUE”.

Далее мы видим инструкцию XOR EAX, EAX, которая точно завершится нулем, независимо от значения в регистре EAX, а значит выставит флаг ZF в 1.

Следовательно, оператор JE выполнит переход и мы пропустим вызов функции MessageBox с текстом “Always FALSE”.

Мы перешли на инструкцию CMP, которая выполняет сравнение с числом 5.

Оператор CMP работает также, как и оператор SUB, т.е. выполняет вычитание, но не сохраняет результат операции. Результат отражается на флаге SF (Signed Flag), который выставляется в 1, если произошла смена знака у первого числа при вычитании.

Давайте попробуем разобраться причем здесь смена знака. Именно так выполняется сравнение двух чисел. Из одного числа вычитается другое и если знак изменился, то первое число было меньше второго.

В данном случае будет выполнено выражение 2 - 5. Результат этой операции -3. Сменился знак первого числа, значит флаг SF будет выставлен в 1.

Далее мы видим оператор JGE (прыгать, если больше или равно).

При операторе JGE переход будет осуществлен, если флаг SF будет выставлен в 0, т.е. только в том случае, когда смена знака не происходит.

В нашем случае перехода не будет и мы выполняем код, который вызывает функцию MessageBox с текстом “tmp < 5”.

Далее с помощью оператора CMP происходит аналогичное сравнение чисел 2 и 5 путем вычитания одного из другого и далее мы видим оператор JLE.

При операторе JLE переход будет осуществлен, если флаг SF выставлен в 1 или флаги SF и OF неравны. Флаг OF - это флаг переполнения, он выставляется в 1, когда происходит целочисленное переполнение.

Результат операции 2 - 5 явно не ноль. Значит смотрим на вторае условие. Флаг SF выставлен в 1, так как произошла смена знака и флаг OF выставлен в 0, так как переполнения не было. Значения флагов отличаются, значит будет осуществлен переход. Мы попадаем на код, который вызывает функцию MessageBox с текстом “tmp <= 5”.

Далее с помощью оператора CMP происходит сравнение 2 и 2. Результатом операции будет выставленный флаг ZF в 1, так 2 - 2 = 0.

Следующий оператор JNZ (прыгать, если не ноль) не будет выполнять переход, так как флаг ZF выставлен в 1. Мы видим окно с надписью “tmp == 2”.

Мы подошли к реализации конструкции switch/case.

С помощью оператор MOV мы кладем значение переменной tmp в нужное место в стеке и выполняем сравнение с помощью оператора CMP.

Первое сравнение осуществляется с 1. Результат сравнения не равен 0, значит флаг ZF выставлен не будет, а это значит, что переход JE не состоится.

Далее происходит сравнение с 2. Результат операции будет равен 0, значит флаг ZF будет выставлен в 1 и переход JE будет выполнен. Мы видим окно с надписью “Case: 2”.

Далее с помощью оператора JMP мы перепрыгиваем код с остальными случаями в конструкции switch/case и переходим к завершению функции.

Обнуляем регистр EAX с помощью оператора XOR. Затем идет эпилог функции и инструкция RETN.